TCP-Aware Backpressure Routing and Scheduling

Hulya Seferoglu, *Member, IEEE* and Eytan Modiano, *Fellow, IEEE*

Abstract—In this work, we explore the performance of backpressure routing and scheduling for TCP flows over wireless networks. TCP and backpressure are not compatible due to a mismatch between the congestion control mechanism of TCP and the queue size based routing and scheduling of the backpressure framework. We propose a TCP-aware backpressure routing and scheduling mechanism that takes into account the behavior of TCP flows. TCP-aware backpressure provides throughput optimality guarantees in the Lyapunov optimization framework, and gracefully combines TCP and backpressure without making any changes to the TCP protocol. The simulation results show that TCP-aware backpressure (i) improves the throughput of TCP flows significantly, (ii) provides fairness across competing TCP flows, and (iii) accommodates both TCP and non-TCP flows in a wireless network, and improves throughput of these flows without hurting fairness.

Index Terms—Transmission Control Protocol (TCP), backpressure routing and scheduling, wireless networks

1 Introduction

The backpressure routing and scheduling paradigm has emerged from the pioneering work [1], [2], which showed that, in wireless networks where nodes route and schedule packets based on queue backlog differences, one can stabilize the queues for any feasible traffic. This seminal idea has generated a lot of research interest. Moreover, it has been shown that backpressure can be combined with flow control to provide utility-optimal operation [3].

The strengths of these techniques have recently increased the interest in practical implementation of the backpressure framework over wireless networks as summarized in Section 6. One important practical problem that remains open, and is the focus of this paper, is the performance of backpressure with Transmission Control Protocol (TCP) flows.

TCP is the dominant transport protocol in the Internet today and is likely to remain so for the foreseeable future. Therefore, it is crucial to exploit throughput improvement potential of backpressure routing and scheduling for TCP flows. However, TCP flows are not compatible with backpressure. Their joint behavior is so detrimental that some flows may never get a chance to transmit. To better illustrate this point, we first discuss the operation of backpressure in the following example.

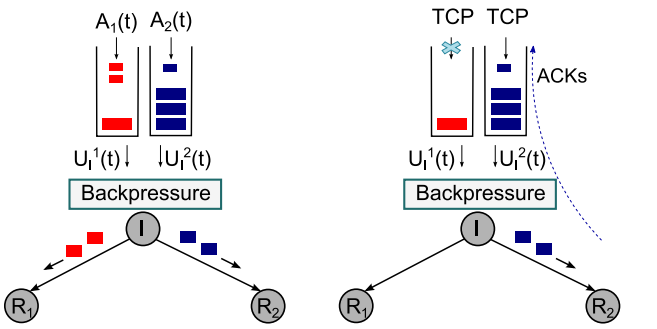
Example 1. Let us consider Fig. 1, which shows an example one-hop downlink topology consisting of a transmitter node I, and two receiver nodes; R_1 and R_2. The two flows; 1 and 2 are destined to R_1 and R_2, respectively. $U_I^1(t)$ and $U_I^2(t)$ are per-flow queue sizes at time t. Let us focus on Fig. 1a. At time t, packets from the two flows arrive according to random arrival rates; $A_1(t)$ and $A_2(t)$, respectively. The packets are stored in per-flow queues. The backpressure scheduling algorithm, also known as max-weight scheduling, determines the queue (hence the flow) from which packets should be transmitted at time t. The decision is based on queue backlog differences, i.e. $U_I^1(t) - U_{R_1}^1(t)$ and $U_I^2(t) - U_{R_2}^2(t)$, where $U_{R_1}^1(t)$ and $U_{R_2}^2(t)$ are per-flow queue sizes at R_1 and R_2, respectively. Since R_1 and R_2 are the destination nodes, the received packets are immediately passed to the higher layers, so $U_{R_1}^1(t) = U_{R_2}^2(t) = 0, \forall t$. Therefore, the scheduling algorithm makes the scheduling decision based on $U_I^1(t)$ and $U_I^2(t)$. In particular, the scheduling decision is $s^* = \arg\max_{s^* \in \{1,2\}} \{U_I^1(t), U_I^2(t)\}$. Note that a packet(s) from flow s^* is transmitted at time t. It was shown in [1], [2] that if the arrivals rates $A_1(t)$ and $A_2(t)$ are inside the stability region, the scheduling algorithm stabilizes the queues. Note that the arrival rates $A_1(t)$ and $A_2(t)$ are independent from the scheduling decisions, i.e. the scheduling decisions do not affect $A_1(t)$ and $A_2(t)$. However, this is not true if the flows are regulated by TCP as explained next.

The fundamental goal of TCP, which applies to all TCP variants, is to achieve as much bandwidth as possible while maintaining some level of long-term rate fairness across competing flows. By their very design, all TCP algorithms (both the widely employed loss-based versions and the delay-based ones) have their own "clock", which relies on end-to-end acknowledgement (ACK) packets. Based on the received ACKs, TCP determines whether and how many packets should be injected into the network by updating its window size.

Example 1—continued. Let us consider Fig. 1b to illustrate the interaction of backpressure and TCP. In Fig. 1b, packet arrivals are controlled by TCP. Let us consider that loss-based TCP flavor, e.g., TCP-Reno or TCP-SACK, is

- *H. Seferoglu is with the Electrical and Computer Engineering Department, University of Illinois at Chicago, Chicago, IL 60607. E-mail: hulya@uic.edu.*
- *E. Modiano is with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139. E-mail: modiano@mit.edu.*

Manuscript received 14 Dec. 2014; revised 25 Aug. 2015; accepted 27 Aug. 2015. Date of publication 14 Sept. 2015; date of current version 1 June 2016. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TMC.2015.2478456



(a) Backpressure with random arrivals with rates $A_1(t)$, $A_2(t)$

(b) Backpressure with TCP arrivals

Fig. 1. Example one-hop downlink topology consisting of a transmitter node I, and two receiver nodes; R_1 and R_2. The two flows; 1 and 2 are destined to R_1 and R_2, respectively. $U_I^1(t)$ and $U_I^2(t)$ are per-flow queue sizes at time t.

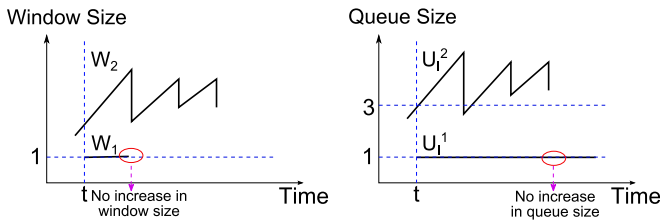


Fig. 2. Sample paths that show the evolution of W_1,W_2 and U_I^1, U_I^2 over time. Note that W_1,W_2 are the congestion window size of the TCP flows, and U_I^1, U_I^2 are the corresponding queue sizes for the example presented in Fig. 1b. Due to backpressure, W_1 does not increase and U_I^1 does not receive or transmit any packets, and its size stays the same; $U_I^1(t) = 1, \forall t$.

employed. Assume that at time t, the TCP congestion window size of the first flow, i.e. $W_1(t)$, is small, e.g., $W_1(t) = 1$ segment, (note that 1-segment window size may be seen at the beginning of a connection or after a re-transmit timeout), while the TCP congestion window size of the second flow is $W_2(t) > 1$ (e.g., it may be the case that flow 2 has been transmitting for some time until t, and it has already increased its window size). As depicted in the figure, the example queue occupancies at time t are $U_I^1(t) = 1$ and $U_I^2(t) = 3$. Since, $U_I^2(t) > U_I^1(t)$, a packet(s) from the second flow is transmitted. R_2 receives the transmitted packet, and passes it to TCP, which generates an ACK, and transmits it back to node I. The TCP source of flow 2 at node I increases its window size after receiving an ACK. Therefore, more packets are passed to $U_I^2(t)$. On the other hand, since $U_I^1(t) < U_I^2(t)$, no packets are transmitted from flow 1. Thus, TCP does not receive any ACKs for the 1st flow, does not increase its window size, and no (or sporadic) new packets are passed to $U_I^1(t)$. Ultimately, the size of $U_I^1(t)$ almost never increases, so no packets are transmitted from flow 1. Possible sample paths showing the evolution of W_1 and W_2 as well as U_I^1 and U_I^2 over time are shown in Fig. 2. As can be seen, the joint behavior of TCP and backpressure is so detrimental that flow 1 does not get any chance to transmit. We confirm this observation via simulations in Section 5.

The incompatibility of backpressure is not limited to the loss-based versions of TCP. The delay-based TCP, i.e., TCP Vegas is also incompatible with backpressure, as TCP-Vegas has its own clock, which relies on end-to-end ACK packets to calculate round-trip-times (RTTs). If some packets are trapped in buffers due to backpressure as in the above example, sporadic or no ACK packets are received. This increases RTTs, and reduces end-to-end rate of TCP Vegas as there is inverse relationship between RTT and rate. Furthermore, backpressure leads to timeouts which reduce the end-to-end rate in both loss-based and delay-based TCP versions, including new TCP versions; TCP-Compound [4] and TCP-Cubic [5].

In this paper, we propose "TCP-aware backpressure" that helps TCP and backpressure operate in harmony. In particular, TCP-aware backpressure takes into account the behavior of TCP flows, and gives transmission opportunity to flows with short queues. This makes all TCP flows transmit their packets, so the TCP clock, which relies on packet transmissions and end-to-end ACKs, continues to operate. Furthermore, the throughput of TCP flows improves by exploiting the performance of the backpressure routing and scheduling. We note that backpressure introduces additional challenges when combined with TCP such as out of order delivery, high jitter in RTTs, and packet losses due to corruption over wireless links. However, these challenges are not specific to backpressure, and exist when a multiple path routing scheme over wireless networks is combined with TCP. We address these challenges by employing network coding as a rateless code (in Section 4). Yet, the main focus of this paper is the incompatibility of TCP and backpressure and developing a TCP-aware backpressure framework. The following are the key contributions of this work:

- We identify the mismatch between TCP and the backpressure framework; i.e. their joint behavior is so detrimental that some flows may never get a chance to transmit. In order to address the mismatch between TCP and backpressure, we develop "TCP-aware backpressure routing and scheduling".
- We show that (i) TCP-aware backpressure routing and scheduling stabilizes queues for any feasible traffic as the classical backpressure [1], [2], (ii) TCP-aware backpressure routing and scheduling provides the same utility-optimal operation guarantee when combined with a flow control algorithm as the classical backpressure [3].
- We provide implementation details and explain how to tune TCP-aware backpressure so that it complies with TCP. Moreover, we combine network coding and TCP-aware backpressure to address the additional challenges such as out of order delivery, packet loss, and jitter. Thanks to employing network coding, which makes TCP flows sequence agnostic (with respect to packet IDs), TCP-aware backpressure fully complies with TCP.
- We develop a TCP-friendly flow control mechanism, which when combined with TCP-aware backpressure, accommodates both TCP and non-TCP flows in a wireless network. In this setup, TCP-aware backpressure improves throughput of both TCP and non-TCP flows and provides fairness across competing flows.

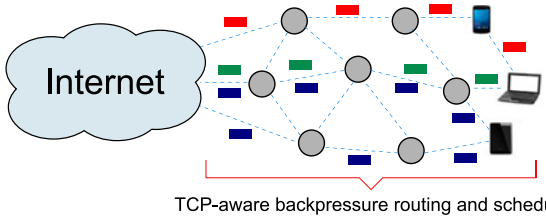


Fig. 3. A general network model that we consider in this paper. A flow may originate from a source in the Internet and traverse multiple hops to reach its destination in a wireless network. An end-to-end TCP connection is set up for each flow. We explore the performance of backpressure for TCP flows in the wireless network.

- We evaluate our schemes in a multi-hop setting, using ns-2 [6]. The simulation results (i) confirm the mismatch of TCP and backpressure, (ii) show that TCP-aware backpressure is compatible with TCP, and significantly improves throughput as compared to existing adaptive routing schemes, (iii) demonstrate that TCP-aware backpressure provides fairness across competing TCP flows, (iv) show that both TCP and non-TCP flows can be accommodated in wireless network with TCP-aware backpressure, and throughput is improved without hurting fairness.

The structure of the rest of the paper is as follows. Section 2 gives an overview of the system model. Section 3 presents TCP-aware backpressure design and analysis. Section 4 presents the implementation details of TCP-aware backpressure as well as its interaction with TCP and non-TCP flows. Section 5 presents simulation results. Section 6 presents related work. Section 7 concludes the paper.

2 System Model

We consider a general network model presented in Fig. 3, where flows may originate from a source in the Internet and traverse multiple hops to reach their destination in a wireless network. An end-to-end TCP connection is set up for each flow. Our goal in this paper is to develop TCP-aware backpressure routing and scheduling algorithms that operate in the wireless network. In this direction, we first develop our algorithms using the Lyapunov optimization framework (which is presented in Section 3) by taking into account the incompatibility of TCP and classical backpressure. In this section, we provide an overview of the system model and assumptions that we use to develop the TCP-aware backpressure. Note that the interaction and implementation of TCP-aware backpressure routing and scheduling with actual TCP flows are presented in Section 4.

Wireless network setup. The wireless network consists of N nodes and L links, where $\mathcal{N}$ is the set of nodes and $\mathcal{L}$ is the set of links in the network. In this setup, each wireless node is able to perform routing and scheduling. Let $\mathcal{S}$ be the set of unicast flows between source-destination pairs in the network. We consider in our formulation and analysis that time is slotted, and t refers to the beginning of slot t.

Channel model. At slot t, $\boldsymbol{C}(t) = \{C_1(t), \ldots, C_l(t), \ldots, C_L(t)\}$ is the channel state vector, where l represents the edges such that $l = (i,j)$, $(i,j) \in \mathcal{L}$ and $i \neq j$. For the sake of analysis, we assume that $C_l(t)$ is the state of link l at time t and takes values from the set $\{ON, OFF\}$ according to a probability distribution which is i.i.d. over time slots. If $C_l(t) = ON$, packets can be transmitted with rate R_l. Otherwise; (i.e., if $C_l(t) = OFF$), no packets are transmitted. Note that our analysis can be extended to more general channel state models [7]. We also consider a Rayleigh fading model in our simulations.

Let $\Gamma_{\boldsymbol{C}(t)}$ denote the set of the link transmission rates feasible at time slot t with channel state $\boldsymbol{C}(t)$, accounting for interference among wireless links. In particular, at every time slot t, the link transmission vector $\boldsymbol{f}(t) = \{f_1(t), \ldots, f_l(t), \ldots f_L(t)\}$ should be constrained such that $\boldsymbol{f}(t) \in \Gamma_{\boldsymbol{C}(t)}$. Hence, $f_l(t)$ takes a value from the set $\{R_l, 0\}$ depending on the channel state and interference among multiple wireless nodes. Also note that $\boldsymbol{f}(t)$ is determined by the scheduling algorithm.

Stability region. Let (λ_s) be the vector of arrival rates $\forall s \in \mathcal{S}$. The network stability region Λ is defined as the closure of all arrival rate vectors that can be stably transmitted in the network, considering all possible routing and scheduling policies [1], [2], [3]. Λ is fixed and depends only on channel statistics and interference.

Flow rates and queue evolution. Each flow $s \in \mathcal{S}$ is generated at its source node according to an arrival process $A_s(t)$, $\forall s \in \mathcal{S}$ at time slot t. The arrivals are i.i.d. over the slots and $\lambda_s = E[A_s(t)]$, $\forall s \in \mathcal{S}$. We assume that $E[A_s(t)]$ and $E[A_s(t)^2]$ are finite. Note that we make i.i.d. arrivals assumption for the purpose of designing and analyzing our algorithms in the Lyapunov optimization framework. This assumption is relaxed in the practical setup when we combine our algorithms with TCP flows in Section 4.

Each node i constructs a per-flow queue $\mathcal{U}_i^s$ for each flow $s \in \mathcal{S}$. The size of the per-flow queue $\mathcal{U}_i^s$ at time t is $U_i^s(t)$. Let $o(s)$ be the source node of flow s. The packets generated according to the arrival process $A_s(t)$ are inserted in the per-flow queue at node $o(s)$, i.e., in $\mathcal{U}_{o(s)}^s$. These queues only store packets from flow $s \in \mathcal{S}$. Each node i such that $i \in \mathcal{N}$ and $i \neq o(s)$, may receive packets from its neighboring nodes and insert them in $\mathcal{U}_i^s$. The transmission rate of flow s from node i to node j is $f_{i,j}^s(t)$. Since the link transmission rate over link (i,j) is $f_{i,j}(t)$ at time t, multiple flows could share the available rate, i.e., $\sum_{s \in \mathcal{S}} f_{i,j}^s(t) \leq f_{i,j}(t)$. Accordingly, at every time slot t, the size of per-flow queues, i.e., $U_i^s(t)$ evolves according to the following dynamics.

$$U_i^s(t+1) \leq \max\left[U_i^s(t) - \sum_{j \in \mathcal{N}} f_{i,j}^s(t), 0\right] + \sum_{j \in \mathcal{N}} f_{j,i}^s(t) + A_s(t) 1_{[i=o(s)]}, \quad (1)$$

where $1_{[i=o(s)]}$ is an indicator function, which is 1 if $i = o(s)$, and 0, otherwise. Note that (1) is inequality, because the number of packets in the queue $U_j^s(t)$ may be less than $f_{j,i}^s(t)$.

3 TCP-Aware Backpressure: Design and Analysis for an Idealized Scenario

In this section, we design and analyze the TCP-aware backpressure scheme for an idealized scenario. In particular, we

provide a stochastic control strategy including routing and scheduling to address the incompatibility between TCP and classical backpressure for an ideal scenario, where arrival rates are i.i.d. Then in Section 4, we will fine-tune our algorithms so that they operate in harmony with real TCP flows, and provide performance analysis of our algorithms with real TCP flows.

TCP-Aware Backpressure:

- *Routing & intra-node scheduling:* The routing & intra-node scheduling part of TCP-aware backpressure determines a flow s from which packets should be transmitted at slot t from node i, as well as the next hop node j to which packets from flow s should be forwarded. The algorithm works as follows.

  Node i observes per-flow queue backlogs in all neighboring nodes at time t, and determines queue backlog difference according to:

$$D^s_{i,j}(t) = \max\{K, U^s_i(t)\} - U^s_j(t), \tag{2}$$

  where K is a non-negative finite constant. Let $l = (i, j)$ s.t. $j \in \mathcal{N}$ and $j \neq i$. The maximum queue backlog difference among all flows over link $l \in \mathcal{L}$ is;

$$D^*_l(t) = \max_{[s \in \mathcal{S} | l \in \mathcal{L}]} \{D^s_l(t)\}. \tag{3}$$

  The flow that maximizes the queue backlog differences over link l is $s^*_l(t)$, which is calculated as

$$s^*_l(t) = \arg\max_{[s \in \mathcal{S} | l \in \mathcal{L}]} \{D^s_l(t)\}. \tag{4}$$

  At time slot t, one or more packets are selected from the queue $\mathcal{U}^{s^*_l(t)}_i$ if $D^*_l(t) > 0$ and $\mathcal{U}^{s^*_l(t)}_i$ has enough packets for transmission. The transmission of the selected packets depends on the channel conditions and interference constraints, and determined by inter-node scheduling.

  Note that TCP-aware backpressure uses queue backlog difference $\max\{K, U^s_i(t)\} - U^s_j(t)$ in (2) instead of $U^s_i(t) - U^s_j(t)$ as in classical backpressure. The advantage of using (2) in TCP-aware backpressure is that node i may select packets from flow s even if the queue size $U^s_i(t)$ is small.[1] This advantage is clarified through an illustrative example later in this section.

- *Inter-node scheduling:* The inter-node scheduling part of TCP-aware backpressure determines link transmission rates considering the link state information and interference constraints.

  Each node i observes the channel state $\boldsymbol{C}(t)$ at time t, and determines a transmission vector $\boldsymbol{f}(t) = \{f_1(t), \ldots, f_l(t), \ldots f_L(t)\}$ by maximizing $\sum_{l \in \mathcal{L}} D^*_l(t)\, f_l(t)$. Note that $\boldsymbol{f}(t)$ should be constrained such that $\boldsymbol{f}(t) \in \Gamma_{\boldsymbol{C}(t)}$, i.e., interference among multiple nodes should be taken into account. The resulting transmission rate $f_l(t)$ is used to transmit packets of flow $s^*_l(t)$ over link l.

Theorem 1. *If channel states are i.i.d. over time slots, the arrival rates $\lambda_s, \forall s \in \mathcal{S}$ are interior to the stability region Λ, and K is a non-negative finite constant, then TCP-aware backpressure stabilizes the network and the total average queue size is bounded.*

Proof. The proof is provided in Appendix A. □

Example 2. Let us consider again Fig. 1b for the operation of TCP-aware backpressure. The example queue occupancies at time t are $U^1_I(t) = 1$ and $U^2_I(t) = 3$. Assume that K in (2) is chosen as $K = 10$. According to TCP-aware backpressure, the scheduling algorithm makes a decision based on the rule $s^* = \arg\max_{s^* \in \{1,2\}} \{\max\{K, U^1_I(t)\}, \max\{K, U^2_I(t)\}\}$. Since $\max_{s \in \{1,2\}} \{K, U^s_I(t)\} = 10$, both flows get equal chance for transmission. Thus, congestion window sizes of both TCP flows evolve in time, and the TCP flows can transmit their packets. We note that one can extend this example for the case; $U^1_I(t) = 7$ and $U^2_I(t) = 12$. In this case, as $K = 10$, packets from the first flow may not get any chance for transmission. Therefore, it is crucial to determine K in practice, which we explain in Section 4.

Note that we propose TCP-aware backpressure; its routing, intra-node scheduling, and inter-node scheduling parts to work with TCP and TCP's end-to-end flow control mechanism. In the next section, we provide implementation details. However, TCP-aware backpressure can also be combined with flow control schemes other than TCP's, which is important for two reasons: (i) it may be possible or preferable to use customized flow control mechanisms instead of TCP's in some systems, (ii) there may be both TCP and non-TCP flows in some systems, where a TCP-friendly flow control mechanism combined with non-TCP flows is crucial to accommodate both TCP and non-TCP flows. We consider the following flow control algorithm, developed in [3], to complement TCP-aware backpressure for non-TCP flows.

The flow control algorithm at node i determines the number of packets from flow s that should be passed to the per-flow queues; $\mathcal{U}^s_i$ at every time slot t according to;

$$\begin{aligned} \max_{\boldsymbol{x}} \; & \sum_{[s \in \mathcal{S} | i = o(s)]} [M g_s(x_s(t)) - U^s_i(t) x_s(t)] \\ \text{s.t.} \; & \sum_{[s \in \mathcal{S} | i = o(s)]} x_s(t) \leq R^{max}_i, \end{aligned} \tag{5}$$

where R^{max}_i is a constant larger than the maximum outgoing rate from node i, M is a positive constant, $x_s(t)$ is the rate of packets that will be inserted to the per-flow queue $\mathcal{U}^s_i$, and $g_s(.)$ is the utility function of flow s.

Theorem 2. *If all flows employ the flow control algorithm in (5) and TCP-aware backpressure (with non-negative finite value of K in (2)), then the admitted flow rates converges to the utility optimal operating point (as the classical backpressure) in the stability region Λ with increasing M.*

1. Note that place-holder backlogs, such as using $U^s_i(t) + K$ instead of $U^s_i(t)$ has been considered in the literature [7]. Although place-holder algorithms are beneficial to improve end-to-end delay, they do not solve the problem we consider in this paper as they do not give transmission opportunity to small queues.

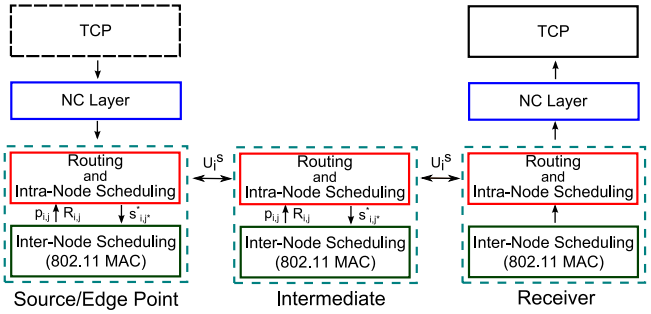


Fig. 4. TCP-aware backpressure operations at edge-points and intermediate nodes. The *inter-node scheduling* and *routing and intra-node scheduling* parts of TCP are implemented on top of 802.11 MAC and in network layers, respectively. The *NC layer* is implemented as a slim layer above the network layer at the edge points. Transport layer, i.e., TCP, only exists if the edge point is a TCP source.

Proof. The proof of Theorem 2 directly follows when Appendix A and drif+penalty approach [3] are combined. We omit the details in this paper. □

4 TCP-Aware Backpressure: Implementation and Analysis for Real TCP Flows

In this section, we first present practical implementation details of TCP-aware backpressure as well as its interaction with different layers in the protocol stack (summarized in Fig. 4). We then discuss how TCP-aware backpressure interacts and performs with real TCP flows, and provide the requirements for TCP-friendly flow control for non-TCP flows when backpressure is employed.

4.1 Implementation

4.1.1 Inter-Node Scheduling

The inter-node scheduling part of TCP-aware backpressure determines which links should be activated at time t. The inter-node scheduling is a hard problem, [9], [10], so its practical implementation is challenging. Therefore, we implement its low complexity version in our system on top of IEEE 802.11 MAC as seen in Fig. 4. The implementation details are as follows.

Each node uses 802.11 MAC to access the wireless medium. When a node i is assigned a channel by the MAC protocol, inter-node scheduling determines the neighboring node that a selected packet should be forwarded to. Let us assume that a packet is selected from flow $s^*_{i,j}(t)$ to be forwarded to node j by the routing and intra-node scheduling algorithm, which we explain later in this section. The next hop that the selected packet should be forwarded is j^* and determined by $j^* = \arg\max_{j\in\mathcal{N}}\{D^*_{i,j}\tilde{R}_{i,j}(1-\tilde{p}_{i,j})\}$, where $\tilde{p}_l$ and $\tilde{R}_l$ are the estimated values of p_l (loss probability) and R_l (link transmission rate) over link $l=(i,j)$, respectively.[2] Then, a packet from flow $s^*_{i,j^*}(t)$, i.e., from the network layer queue $U_i^{s^*_{i,j^*}(t)}$, is removed and passed to the MAC layer for transmission. The MAC layer transmits the packet to node j^*.

2. $\tilde{p}_l$ is calculated as one minus the ratio of successfully transmitted packets over all transmitted packets during a time interval T on link l. $\tilde{R}_l$ is calculated as the average of the recent (over an interval) link rates over link l.

4.1.2 Routing and Intra-Node Scheduling

This algorithm determines the next hop(s) to which packets should be forwarded, and the packets that should be transmitted.

We construct per-flow queues, i.e., $\mathcal{U}_i^s$, at the network layer,[3] where the routing and intra-node scheduling algorithm operates as seen in Fig. 4. The algorithm requires each node to know the queue size of their neighbors. To achieve this, each node i transmits a message containing the size of its per-flow queue sizes; U_i^s at time t. These messages are piggy-backed to data packets. If there is no data transmission for some time duration, our algorithm uses independent control packets to exchange the queue size information. The transmitted message is overheard by all nodes in the neighborhood. The queue size information is extracted from the overheard messages and recorded for future decisions.

At node i at time t, the queue backlog difference is calculated according to (2). Note that, although the algorithm exactly knows $U_i^s(t)$ at time t, it is difficult to exactly know $U_j^s(t)$ at time t. Therefore, the most recent report (until time t) of the size of $\mathcal{U}_j^s$ is used instead of $U_j^s(t)$. When a transmission opportunity for link (i,j) arises using inter-node scheduling algorithm, a packet from flow $s^*_{i,j}(t)$ is selected and passed to the MAC layer for transmission.

4.1.3 Network Coding

Out of order delivery, high jitter in RTTs, and packet losses over wireless links are among the challenges when backpressure and TCP are combined. We address these challenges by employing network coding in the form of rateless codes [13], [14], [15]. This is an effective solution thanks to the properties of network coding such as masking wireless losses and making packets sequence agnostic in terms of packet IDs. We summarize our implementation in the following.

We implement the generation based network coding [16] at the edge points of the wireless network (e.g., access point, base station, proxy, or TCP source itself) as a slim network coding layer (NC layer) above the network layer as shown in Fig. 4. Note that we do not make any updates to TCP, which makes our approach amenable to practical deployment.

The NC layer at the edge point receives and packetizes the data stream into packets $\eta^s_1, \eta^s_2, \ldots$ of flow $s \in \mathcal{S}$. The stream of packets are divided into blocks of size H_s, which is set to TCP congestion window size (or its average). The packets within the same block are linearly combined (assuming large enough field size) to generate H_s network coded packets; $a^s_1 = \alpha_{1,1}\eta^s_1$, $a^s_2 = \alpha_{2,1}\eta^s_1 + \alpha_{2,2}\eta^s_2, \ldots, a^s_{H_s} = \alpha_{H_s,1}\eta^s_1 + \cdots + \alpha_{H_s,H_s}\eta^s_{H_s}$, where $\alpha_{i,j}, \forall i,j$ are network coding coefficients from a finite field. Note that network coded packets are generated incrementally to avoid coding delay [16], [15]. The NC layer adds network coding header

3. Note that constructing per-flow queues at each node may not be feasible in some systems. However, this aspect is orthogonal to the focus of this paper, and the techniques developed in the literature [11], [12] to address this problem is complementary to our TCP-aware backpressure framework.

including block ID, packet ID, block size, and coding coefficients. The network coded packets are routed and scheduled by TCP-aware backpressure.

At the receiver node, when the NC layer receives a packet from a new block, it considers the received packet as the first packet in the block. It generates an ACK, sends the ACK back to the NC layer at the edge point, which matches this ACK to packet η_1, converts this ACK to η_1's ACK, and transmits the ACK information to the TCP source. Similarly, ACKs are generated at the receiver side for the second, third, etc. received packets. As long as the NC layer at the receiver transmits ACKs, the TCP clock moves, and the window continues to advance.

The NC layer stores the received network coded packets in a buffer. When the last packet from a block is received, packets are decoded and passed to the application layer. If some packets are lost in the wireless network, the receiver side NC layer makes a request with the block ID and the number of missing packets, and the edge point side NC layer generates additional network coded packets from the requested block, and sends to the receiver. Note that the missing packet IDs are not mentioned in the request, since the network coding makes the packets sequence agnostic in terms of packet IDs.

Network coding makes packets sequence agnostic, which solves out of order delivery problem and eliminates jitter. Network coding also corrects packet losses in the wireless network as explained above. We explain how our system and NC layer reacts to congestion-based losses later in Section 4.2.1.

4.1.4 Selection of K

TCP-aware backpressure uses queue backlog difference in (2), which depends on K, to make routing and scheduling decisions. As noted in Section 3, the selection of K is crucial in practice to make TCP and backpressure fully comply.

In particular, if K is selected too small, the number of packets that are trapped in the buffers, i.e., the number of packets that do not get transmission opportunity, increases. This reduces TCP throughput. On the other hand, if K is too large, TCP-aware backpressure may not exploit the throughput improvement benefit of backpressure routing and scheduling as the ability of identifying good routing and scheduling policies reduces with large K values.

Our intuition is that flows passing through node i, i.e., $s \in \mathcal{S}_i$, should share the available buffer fairly. Assume that B_i is the available buffer size at node i. In order to give transmission opportunity to all TCP flows and provide some level of fairness across the competing TCP flows, we set $K = B_i/|\mathcal{S}_i|$ at node i. In this setting, if per-flow queue sizes are smaller than K, it is likely that packets from all TCP flows are transmitted. On the other hand, if some per-flow queue sizes are larger than K, packets from the flows with smaller queue sizes may still be trapped in the buffers. However, in this case, since the total buffer occupancy is large, buffer overflow probability at the source/edge node increases. Upon buffer overflow, the TCP flow with larger queue size reduces its rate (since upon congestion a packet from the largest per-flow queue is dropped). This reduces the queue sizes, and packets from all flows could be transmitted again.

Example 2—continued: Let us consider again Fig. 1b. If the queue occupancies are $U_I^1(t) = 7$, $U_I^2(t) = 12$, and $K = 10$, packets only from the second flow are transmitted. Since $K = 10$ and we set $K = B_I/|\mathcal{S}_I|$, and $|\mathcal{S}_I| = 2$, the buffer size is $B_I = 20$. The total queue occupancy is $U_I^1(t) + U_I^2(t) = 19$. This means that the buffer at node I is about to overflow, which will lead to back-off for the second flow (since a packet from the largest queue will be dropped). Thus, the TCP rate and queue size of the second flow will reduce, and the first flow will get transmission opportunity.

We have observed through simulations that TCP-aware backpressure, when K is set to $B_i/|\mathcal{S}_i|$, significantly reduces the number of the trapped packets in the buffers. Yet, a few packets may still be trapped. Such packets are easily masked thanks to error correction capabilities of network coding. Note that network coding does not help if large number of packets are trapped in the buffers (e.g., when K is selected too small), as large number of trapped packets increases end-to-end delay too much, which leads to multiple timeouts and reduces TCP throughput.

4.2 Interaction with TCP Flows

4.2.1 Congestion Control

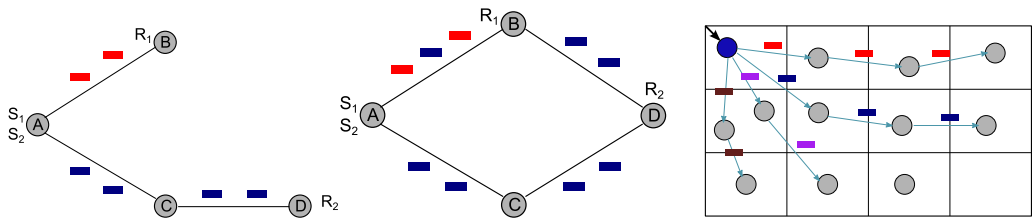
Now, let us consider the interaction of TCP congestion control and TCP-aware backpressure. When TCP-aware backpressure is employed, using the similar approach in [17], [18], we find the steady state TCP throughput for flow s as; $x_s^2 = \frac{(1-q^s_{o(s)})}{T_s^3 q^s_{o(s)}}$, where $q^s_{o(s)}$ is the buffer overflow probability at the TCP source/edge node $o(s)$, and T_s is constant RTT.[4] (The details of our analysis is provided in Appendix B.) Note that the steady state TCP throughput depends on the buffer overflow probability only at the source/edge node[5] which is different from [17], [18] where TCP throughput depends on the buffer overflow probability over all nodes over the path of TCP flow. Thus, in our implementation, if the buffer at the source/edge node is congested, a packet from the flow which has the largest queue size is dropped. This congestion-based loss information is passed to the NC layer. The NC layer creates a loss event by not masking the dropped packet so that TCP can detect the congestion-based loss event and back-off.

4.2.2 Performance of TCP-Aware Backpressure with Real TCP Flows

In Section 3, we showed that our TCP-aware backpressure algorithm provides stability and optimality guarantees when arrival rates are i.i.d. In this section, our goal is to analyze the performance of TCP-aware backpressure with real TCP flows using a fluid-based approach as in [18].

4. The constant RTT is a common assumption in classical TCP analysis [17], [18], and also valid in our setup thanks to employing network coding, which reduces jitter in RTT and makes constant RTT assumption valid. A detailed discussion is provided in Appendix B.

5. Note that buffer overflows only occurs at the source node, not in the other (intermediate) nodes in TCP-aware backpressure. The reason is that our algorithm makes transmission decisions based on queue sizes. I.e., a node does not transmit packets to a next hop if there is possibility of overflow in the next hop.



(a) Tree topology (b) Diamond topology (c) Grid topology

Fig. 5. Topologies used in simulations.

In particular, when TCP-aware backpressure for routing and scheduling and a TCP source (specifically TCP-Reno) are used together, the equilibrium point approaches the solution of the optimization problem: $\max_{\boldsymbol{x},\boldsymbol{f}} \sum_{s\in\mathcal{S}} g_s(x_s)$ when $g_s(x_s) = \frac{1}{T_s\sqrt{\beta}} \tan^{-1}(\sqrt{\beta}T_s)x_s$ subject to $\sum_{j\in\mathcal{N}} f^s_{i,j} - \sum_{j\in\mathcal{N}} f^s_{j,i} = x_s 1_{[i=o(s)]}, \forall i \in \mathcal{N}, s \in \mathcal{S}$ and $\boldsymbol{f} \in \Gamma$, where Γ is the stability region of the network. The details of the analysis is provided in Appendix C.

4.3 Interaction with Non-TCP Flows and TCP-Friendly Flow Control

As mentioned in Section 3, there may be both TCP and non-TCP flows in a network, and non-TCP flows should be controlled in a TCP-friendly manner so that TCP flows could survive when non-TCP flows are active. Therefore, we presented a flow control algorithm in (5) to control non-TCP flows. Yet, the parameters of the flow control algorithm, in particular M, should be selected so that these non-TCP flows could be TCP-friendly. This is the focus of this section.

In this section, we consider (5) in a fluid from. In this case, (5) is expressed as; $\max_{\boldsymbol{x}} \sum_{[s\in\mathcal{S}|i=o(s)]} [Mg_s(x_s) - U^s_i x_s]$. Let us first consider that the utility function is $g_s(x_s) = log(x_s)$. In this case, as $\max_{\boldsymbol{x}} \sum_{[s\in\mathcal{S}|i=o(s)]} [Mlog(x_s) - U^s_i x_s]$ is an unconstrained convex optimization problem, the steady state non-TCP flow rate is calculated as $x_s = M/U^s_{o(s)}$. To provide TCP friendliness, non-TCP flow rate should be equal to TCP flow rate (which is calculated as $x^2_s = \frac{(1-q^s_{o(s)})}{T^3_s \beta q^s_{o(s)}}$ in Appendix B) [33]. From this equality we can find M as

$$M = \frac{U^s_{o(s)}}{\sqrt{T^3_s \beta}} \sqrt{\frac{1-q_{o(s)}}{q_{o(s)}}}. \tag{6}$$

Now, let us consider the general utility function $g_s(x_s) = \frac{x_s^{1-\alpha}}{1-\alpha}$, for $\alpha > 0$, which provides different levels of fairness for different α values [34]; e.g., when $\alpha \to 1$, the utility function becomes $g_s(x_s) \to log(x_s)$ and provides proportional rate fairness, and when $\alpha \to \infty$, the utility function $g_s(x_s)$ provides max-min fairness. For this general utility function, we can calculate M as $M = U^s_{o(s)} x^\alpha_s$. To provide TCP-friendliness, M should be

$$M = U^s_{o(s)} \left[\frac{1-q_{o(s)}}{T^3_s \beta q_{o(s)}}\right]^{\alpha/2}. \tag{7}$$

Using the similar approach, we can develop rules for M for any convex utility function.

5 Performance Evaluation

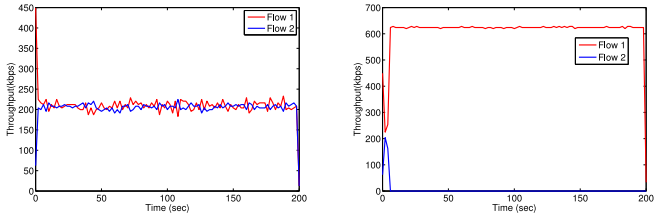
We simulate our scheme, TCP-aware backpressure (TCP-aware BP) as well as classical backpressure (classical BP), in ns-2 [6]. The simulation results; (i) confirm the mismatch of TCP and classical BP, (ii) show that TCP-aware BP is compatible with TCP, and significantly improves throughput as compared to existing routing schemes such as Ad-hoc On-Demand Distance Vector (AODV) [19], (iii) demonstrate that TCP-aware BP provides fairness across competing TCP flows. Next, we present the simulator setup and results in detail.

5.1 Simulation Setup

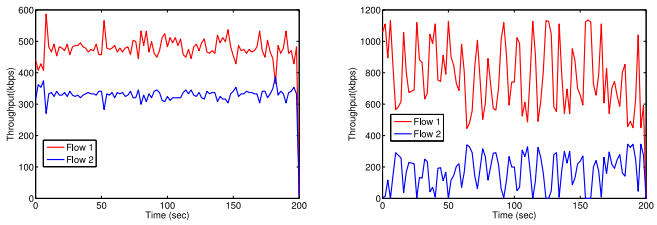
We consider three topologies: a tree topology, a diamond topology, and a grid topology shown in Fig. 5. The nodes are placed over $500 \times 500\,\mathrm{m}$ terrain, and S_1, S_2 and R_1, R_2 are possible source-receiver pairs in the tree and diamond topologies. In the grid topology, 4×3 cells are placed over a $800 \times 600\,\mathrm{m}$ terrain. A gateway, which is connected to the Internet, passes flows to nodes. Each node communicates with other nodes in its cell or neighboring cells, and there are 12 nodes randomly placed in the cells.

We consider FTP/TCP traffic, and employ TCP-SACK and TCP-Vegas in our simulations. TCP flows start at random times within the first $5\,\mathrm{sec}$ of the simulation and are on until the end of the simulation which is $200\,\mathrm{sec}$. IEEE 802.11b is used in the MAC layer. In terms of wireless channel, we simulated the two-ray path loss model and a Rayleigh fading channel with average loss rates $0, 20, 30, 40, 50$ percent. Channel capacity is $2\,\mathrm{Mbps}$, the buffer size at each node is set to 100 packets, packet sizes are set to $1,000\,\mathrm{B}$. We have repeated each $200\,\mathrm{sec}$ simulation for 10 seeds.

The following is the summary of our evaluation results we present in the next section. (i) We present the performance of TCP-aware BP as compared to the classical BP, and demonstrate that TCP-aware BP supports all TCP flows in the system while some TCP flows do not survive in the classical BP. (ii) We present the performance of TCP-aware BP as compared to AODV, and demonstrate that TCP-aware BP improves throughput significantly as compared to AODV thanks to exploiting better routes. (iii) We present the performance of TCP-aware BP while accommodating both TCP and non-TCP flows, and demonstrate that TCP-aware BP can accommodate both TCP and non-TCP flows, which is not possible in AODV. Note that for fair comparison, we employ the network coding mechanism explained in Section 4 in the classical BP as well as in AODV. The comparisons are in terms of per-flow and total transport level



(a) TCP-Aware BP with TCP-SACK (b) BP with TCP-SACK



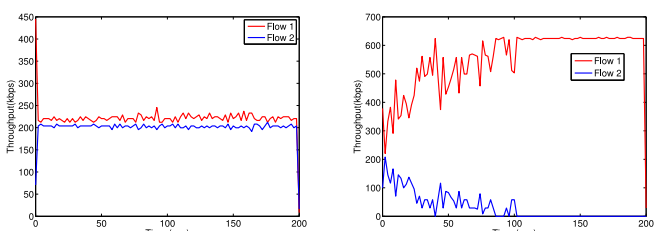
(c) TCP-Aware BP with TCP-Vegas (d) BP with TCP-Vegas

Fig. 6. Throughput versus time in the tree topology for TCP-SACK and TCP-Vegas. There are two flows; Flow 1 is transmitted from node A to node B, and Flow 2 is transmitted from node A to node D. The links are not lossy.

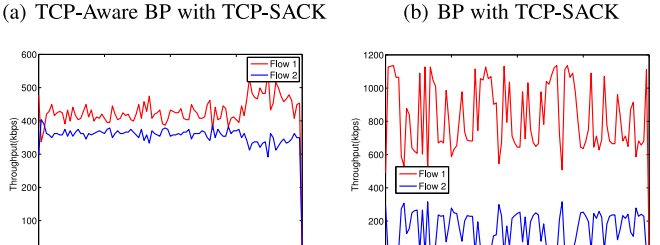
throughput (added over all flows) as well as fairness. For the fairness calculation, we use Jain's fairness index [20]: $F = \frac{(\sum_{s\in\mathcal{S}} \bar{x}_s)^2}{|\mathcal{S}|(\sum_{s\in\mathcal{S}} (\bar{x}_s)^2)}$, where $\mathcal{S}$ is the set of flows and $\bar{x}_s$ is the average throughput of flow s.

5.2 Simulation Results

Performance of TCP-aware BP as compared to BP: Fig. 6 shows throughput versus time for TCP-aware BP and classical BP for the tree topology shown in Fig. 5a. There are two flows; Flow 1 is transmitted from node A to node B, and Flow 2 is transmitted from node A to node D. The links are not lossy. Figs. 6a and 6b are the results for TCP-SACK, while Figs. 6c and 6d are for TCP-Vegas. Fig. 6b shows that while Flow 1 is able to transmit, Flow 2 does not get any chance for transmission in classical BP due to the mismatch between congestion window size update mechanism of TCP and queue size-based routing and scheduling of backpressure. On the other hand, in TCP-aware BP, both flows get chance for transmission. In particular, Flow 1 and Flow 2 achieves average throughput of 205.76 and 203.36 kbps in Fig. 6, respectively. Figs. 6c and 6d show throughput versus time graphs of TCP-aware BP and classical BP for TCP-Vegas. Although classical BP performs better in TCP-Vegas than in TCP-SACK due to the delay-based mechanism of TCP-Vegas, its performance is still quite poor as the throughput of Flow 2 frequently goes to 0 as seen in Fig. 6d. On the other hand, TCP-aware BP improves throughput of both flows as seen in Fig. 6c, where Flow 1 and Flow 2 achieve 469.36 and 324.64 kbps, respectively. Similar results are presented in Fig. 7 for the diamond topology in Fig. 5b.

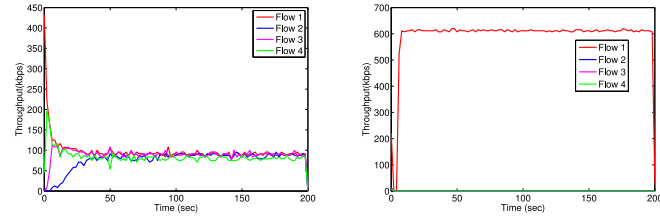


(a) TCP-Aware BP with TCP-SACK (b) BP with TCP-SACK

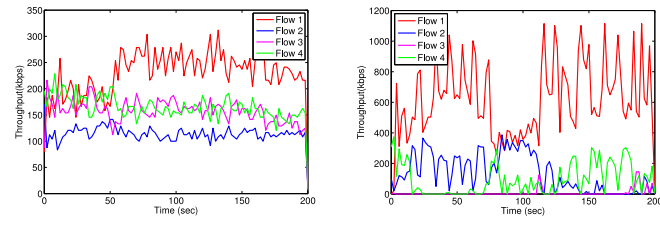


(c) TCP-Aware BP with TCP-Vegas (d) BP with TCP-Vegas

Fig. 7. Throughput versus time in the diamond topology for TCP-SACK and TCP-Vegas. There are two flows; Flow 1 is transmitted from node A to node B, and Flow 2 is transmitted from node A to node D. The links are not lossy.



(a) TCP-Aware BP with TCP-SACK (b) BP with TCP-SACK

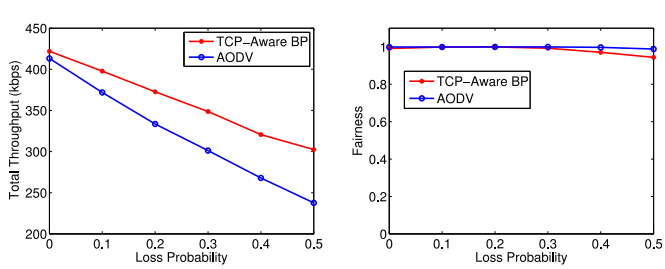


(c) TCP-Aware BP with TCP-Vegas (d) BP with TCP-Vegas

Fig. 8. Throughput versus time in the grid topology for TCP-SACK and TCP-Vegas. There are four flows and the links are not lossy.

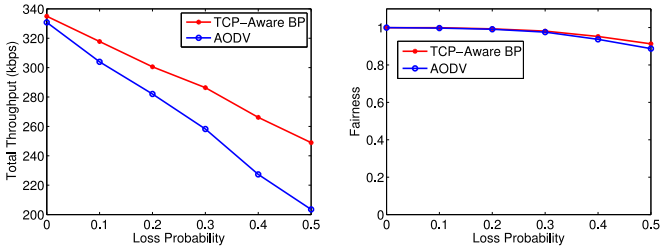
Let us consider the grid topology shown in Fig. 5c. Four flows are transmitted from the gateway to four distinct nodes, which are randomly chosen. Half of the links, chosen at random, are lossy with loss probability ranging between $0-0.5$. Fig. 8 shows throughput versus time graphs for TCP-aware BP and classical BP. It is seen that all four flows could survive in TCP-aware BP for both TCP-SACK and TCP-Vegas, while one or more flows do not survive in classical BP.

Performance of TCP-aware BP as compared to AODV: Fig. 9 demonstrates throughput and fairness versus average loss



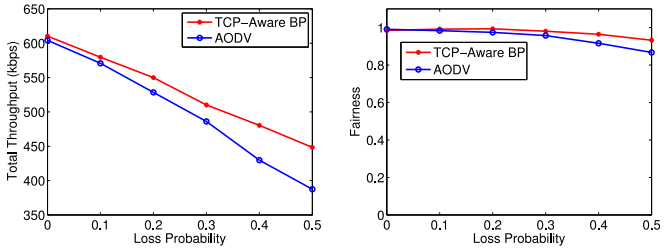
(a) Throughput (b) Fairness

Fig. 9. Throughput and fairness versus average packet loss rate for TCP-aware BP and AODV in the diamond topology. There are two TCP flows transmitted from node A to B (Flow 1) and A to D (Flow 2). The link $A-B$ is a lossy link. The version of TCP is TCP-SACK.



(a) Throughput (b) Fairness

Fig. 10. Throughput and fairness versus average packet loss rate for TCP-aware BP and AODV in the grid topology. There are four TCP flows transmitted from the gateway to four distinct nodes. Half of the links are lossy. The version of TCP is TCP-SACK.



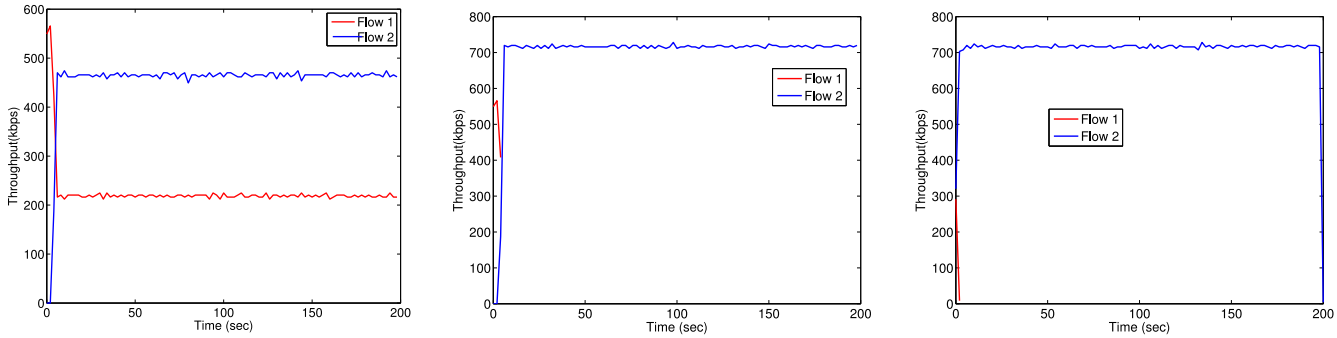
(a) Throughput (b) Fairness

Fig. 11. Throughput and fairness versus average packet loss rate for TCP-aware BP and AODV in the grid topology. There are four TCP flows transmitted from the gateway to four distinct nodes. Half of the links are lossy. The version of TCP is TCP-Vegas.

rate results of TCP-aware BP and AODV in the diamond topology shown in Fig. 5b. There are two flows transmitted from node A to B (Flow 1) and A to D (Flow 2). The link $A-B$ is a lossy link. The version of TCP is TCP-SACK. Fig. 9a shows that TCP-aware BP improves throughput significantly as compared to AODV thanks to adaptive routing and scheduling. The throughput improvement of TCP-aware BP as compared to AODV increases as loss probability increases thanks to loss-aware routing and scheduling mechanism of TCP-aware BP. Moreover, Fig. 9b shows that the fairness index is close to $F=1$ (note that $F=1$ is the highest possible fairness index) when TCP-aware BP is employed. This means that both TCP flows are able to survive in TCP-aware BP. Note that the fairness index of TCP-aware BP is 0.94, while the fairness index of AODV is 0.98 when the packet loss probability is 0.5. This is due to the fact that TCP-aware BP exploits loss-free links better, and slightly favors the flows transmitted over such links. However, the throughput improvement of both flows as compared to AODV is higher. In particular, TCP-aware BP improves throughput as compared to AODV by 10 and 40 percent for the first and second flows, respectively. These results confirm the compatibility of TCP and TCP-aware BP.

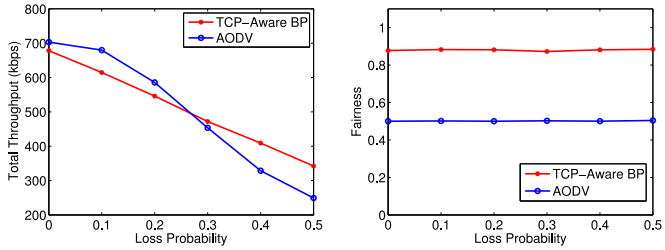
Fig. 10 shows throughput and fairness versus average loss probability results for TCP-aware BP and AODV for TCP-SACK for the grid topology shown in Fig. 5c. In this topology, four flows are transmitted from the gateway to four distinct nodes, which are randomly chosen. Half of the links, chosen at random, are lossy with loss probability ranging between $0-0.5$. TCP-aware BP improves throughput significantly as compared to AODV without violating fairness. Fig. 11 shows that TCP-aware BP improves throughput significantly as compared to AODV when TCP-Vegas is employed. This shows the effectiveness of our scheme in delay-based TCP versions.

Performance of TCP-aware BP while accommodating both TCP and non-TCP flows. As mentioned in Sections 3 and 4.3, there may be both TCP and non-TCP flows in the system, and non-TCP flows should be controlled in a TCP-friendly manner so that TCP flows could survive when non-TCP flows are on. Therefore, a flow control algorithm is presented in (5) for non-TCP flows. Now, we evaluate this scenario in the diamond topology with two flows. Flow 1 is a TCP flow (TCP-SACK) transmitted from node A to node B, and Flow 2 is a non-TCP flow transmitted from node A to node D. In our TCP-aware BP framework, the non-TCP flow is regulated by (5). We consider $\log$ utility function, i.e., $g(x_s(t)) = log(x_s(t)),\ \forall t, s \in \mathcal{S}$. The implementation details including TCP-friendly parameter selection are provided in Section 4.3. Fig. 12 shows throughput versus time graph of TCP-aware BP, classical BP, and AODV. The TCP flow does not survive in classical BP as packets are trapped in the buffers. It does not survive with AODV as well, because uncontrolled non-TCP flows (i.e., UDP flows) occupy buffers and TCP packets are constantly dropped from the buffers, which reduces TCP throughput. Yet, both TCP and non-TCP flows survive together in TCP-aware BP thanks to TCP-aware routing and scheduling, and TCP-friendly flow control for non-TCP flows. Fig. 13 shows the throughput improvement performance of TCP-aware BP as compared to AODV in the same setup for different packet loss probabilities. At low loss probabilities, although the



(a) TCP-Aware BP (b) Classical BP (c) AODV

Fig. 12. Throughput versus time in the diamond topology for TCP-SACK. There are two flows; Flow 1 is a TCP flow, transmitted from node A to node B, and Flow 2 is a non-TCP flow, transmitted from node A to node D. The links are not lossy.



(a) Throughput (b) Fairness

Fig. 13. Throughput and fairness versus average packet loss rate for TCP-aware BP and AODV in the diamond topology. There are two flows transmitted from node A to B (Flow 1, i.e., TCP flow) and A to D (Flow 2, i.e., non-TCP flow). The link $A-B$ is a lossy link. The version of TCP is TCP-SACK.

throughput of AODV is better than TCP-aware BP, the fairness graph (and Fig. 12 for no-loss) shows that the fairness of AODV is very low, which means that the TCP flow does not survive. At higher loss probabilities, TCP-aware BP is better than AODV thanks to choosing better routes and schedules as compared to AODV.

6 Related Work

Backpressure, a routing and scheduling framework over communication networks [1], [2] has generated a lot of research interest [7], mainly in wireless ad-hoc networks. It has also been shown that backpressure can be combined with flow control to provide utility-optimal operation guarantee [3], [21].

The strengths of backpressure have recently increased the interest on practical implementation of backpressure over wireless networks. Backpressure has been implemented over sensor networks [22] and wireless multi-hop networks [23]. The multi-receiver diversity has been explored in wireless networks using backpressure in [24]. The 802.11 compliant version of enhanced backpressure is evaluated in [25]. Backpressure routing and rate control for intermittently connected networks was developed in [26].

Backpressure routing and (max-weight) scheduling with TCP over wireless has been considered in the literature. At the link layer, [27], [28], propose, analyze, and evaluate link layer backpressure-based implementations with queue prioritization and congestion window size adjustment. The interaction of TCP with backpressure in [27] and [28] is handled by updating the TCP congestion window evolution mechanism. In particular, if the queue size (at the TCP source) increases, the window size is reduced, otherwise, the window size is increased. Multi-path TCP scheme is implemented over wireless mesh networks [29] for routing and scheduling packets using a backpressure based heuristic, which avoids incompatibility with TCP. Max-weight scheduling is updated in [30] to make decisions based only on MAC level queue size information. Although [30] considers window based flow control mechanism similar to TCP, it does not consider existing TCP flavors. The main differences in our work are: (i) we consider the incompatibility of TCP with backpressure, and develop TCP-aware backpressure framework to address the incompatibilities, (ii) TCP-aware backpressure provides the same stability and utility-optimal operation guarantees as classical backpressure, (iii) we do not make any changes at the TCP source, (iv) we employ network coding to gracefully combine TCP and TCP-aware backpressure.

Maximum weight matching (MWM) is a switch scheduling algorithm and has similar properties as the max-weight scheduling algorithm and backpressure. Similar to the backpressure, there is incompatibility between TCP and MWM [31], [32]. Yet, we consider backpressure routing and scheduling over wireless networks rather than switch scheduling, and we take a holistic approach to address this problem; i.e., we propose TCP-aware backpressure to make TCP and backpressure compatible.

The delay-based routing and scheduling algorithms can be also be utilized with TCP flows. However, the delay-based solutions have two disadvantages in this setup. First, providing performance guarantees for delay-based algorithms are quite involved and is an open problem for general networks [35]. Furthermore, they require clock synchronization, which is quite difficult in practice. As compared to this line of work, we propose TCP-aware backpressure with provable performance guarantees. Also, since TCP-aware backpressure does not introduce complications such as clock synchronization, or updating TCP, it is very suitable for practical deployment.

7 Conclusion

We proposed TCP-aware backpressure routing and scheduling to address the incompatibility of TCP and backpressure while exploiting the performance of backpressure routing and scheduling over wireless networks. TCP-aware backpressure is developed by taking into account the behavior of TCP flows, and gracefully combines TCP and backpressure without making any changes to the TCP protocol. Simulations in ns-2 demonstrate that TCP-aware backpressure improves throughput of TCP flows significantly and provides fairness across competing TCP flows. Furthermore, the simulation results show that TCP-aware backpressure accommodates both TCP and non-TCP flows in a wireless network, and improves throughput of these flows without hurting fairness.

Appendix A
Proof of Theorem 1

The proof of Theorem 1 directly follows from (γ, ξ) (or C-) approximation in [8], [7]. We provide the proof in this section for completeness. Let $f^{*}_{i,j}{}^{s}(t)$ be the optimal decision when $D^{s}_{i,j}(t) = U^{s}_{i}(t) - U^{s}_{j}(t)$ in (2) (note that this is the classical backpressure), while $f^{s}_{i,j}(t)$ be the decision when $D^{s}_{i,j}(t) = \max\{K, U^{s}_{i}(t)\} - U^{s}_{j}(t)$. If the following inequality holds, the policy that makes decision based on $D^{s}_{i,j}(t) = \max\{K, U^{s}_{i}(t)\} - U^{s}_{j}(t)$ stabilizes the queues:

$$\sum_{(i,j)\in\mathcal{L}}\sum_{s\in\mathcal{S}} f^{s}_{i,j}(t)(U^{s}_{i}(t) - U^{s}_{j}(t)) \geq \sum_{(i,j)\in\mathcal{L}}\sum_{s\in\mathcal{S}} f^{*}_{i,j}{}^{s}(t)(U^{s}_{i}(t), - U^{s}_{j}(t)) - Z, \tag{8}$$

where Z is a finite constant. Let us first show that (8) holds. Consider the inequality; $\sum_{(i,j)\in\mathcal{L}}\sum_{s\in\mathcal{S}}(U^{s}_{i}(t) - U^{s}_{j}(t))\, f^{*}_{i,j}{}^{s}(t) \leq$

$\sum_{(i,j)\in\mathcal{L}}\sum_{s\in\mathcal{S}}\max\{K,U_i^s(t)\}\,f_{i,j}^{*\,s}(t)$. This inequality holds, because $\max\{K,U_i^s(t)\}\geq U_i^s(t)$. Also, considering that $f_{i,j}^s(t)$ is the optimal decision when the backlog difference is $\max\{K,U_i^s(t)\}-U_j^s(t)$, the term $(\max\{K,\ U_i^s(t)\}-U_j^s(t))$ $f_{i,j}^s(t)$ should be greater than $(\max\{K,U_i^s(t)\}-U_j^s(t))\ f_{i,j}^{*\,s}(t)$. Therefore, the inequality is expressed as; $\sum_{(i,j)\in\mathcal{L}}$ $\sum_{s\in\mathcal{S}}(U_i^s(t)-U_j^s(t))\,f_{i,j}^{*\,s}(t)\leq\sum_{(i,j)\in\mathcal{L}}\sum_{s\in\mathcal{S}}\max\{K,U_i^s(t)\}$ $f_{i,j}^s(t)$. By adding and removing terms, and noting that $\max\{K,U_i^s(t)\}-U_i^s(t)\leq K$ and $f_{i,j}^s(t)\leq F_{\max}$ such that $F_{\max}\geq R_{i,j}$, the following holds;

$$\sum_{(i,j)\in\mathcal{L}}\sum_{s\in\mathcal{S}}(U_i^s(t)-U_j^s(t))f_{i,j}^s(t) \geq \sum_{(i,j)\in\mathcal{L}}\sum_{s\in\mathcal{S}}(U_i^s(t)-U_j^s(t))f_{i,j}^{*\,s}(t)-|\mathcal{L}||\mathcal{S}|KF_{\max}, \tag{9}$$

(9) verifies that (8) holds considering that $Z=|\mathcal{L}|\ |\mathcal{S}|\ K\ F_{\max}$. Note that (9) is equivalent to;

$$\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}U_i^s(t)\left(\sum_{j\in\mathcal{N}}f_{i,j}^s(t)-\sum_{j\in\mathcal{N}}f_{j,i}^s(t)\right) \geq\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}U_i^s(t)\left(\sum_{j\in\mathcal{N}}f_{i,j}^{*\,s}(t)-\sum_{j\in\mathcal{N}}f_{j,i}^{*\,s}(t)\right)-|\mathcal{L}||\mathcal{S}|KF_{\max}. \tag{10}$$

Now, let us define the Lyapunov function as; $L(\boldsymbol{U}(t))=\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}U_i^s(t)^2$, where $\boldsymbol{U}(t)=\{U_i^s(t)\}_{i\in\mathcal{N},s\in\mathcal{S}}$ and $U_i^s(t)$ evolves according to (1). Let the Lyapunov drift be $\Delta(\boldsymbol{U}(t))=E[L(\boldsymbol{U}(t+1))-L(\boldsymbol{U}(t))|\boldsymbol{U}(t)]$, which is equal to $\Delta(\boldsymbol{U}(t))\leq E[\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}(U_i^s(t+1))^2-\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}\ (U_i^s(t))^2|$ $\boldsymbol{U}(t)]$. Using the fact that $(\max(Q-b,0)+A)^2\leq Q^2+A^2+$ $b^2+2Q(A-b)$, we have; $\Delta(\boldsymbol{U}(t))\leq E[\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}\ (\sum_{j\in\mathcal{N}}$ $f_{i,j}^s(t))^2+\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}(\sum_{j\in\mathcal{N}}f_{i,j}^s(t)+A_s(t)1_{[i=o(s)]})^2+\sum_{i\in\mathcal{N}}$ $\sum_{s\in\mathcal{S}}2U_i^s(t)(\sum_{j\in\mathcal{N}}f_{j,i}^s(t)-\sum_{j\in\mathcal{N}}f_{i,j}^s(t))+\sum_{i\in\mathcal{N}}\ \sum_{s\in\mathcal{S}}2U_i^s(t)$ $A_s(t)1_{[i=o(s)]}|\boldsymbol{U}(t)]$. Noting that there always exist a finite constant B such that $B\geq E[\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}(\sum_{j\in\mathcal{N}}\ f_{i,j}^s(t))^2+$ $\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}(\sum_{j\in\mathcal{N}}\ f_{i,j}^s(t)+A_s(t)1_{[i=o(s)]})^2|\boldsymbol{U}(t)]$, we have;

$$\Delta(\boldsymbol{U}(t))\leq B-2E\Bigg[\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\sum_{s\in\mathcal{S}}U_i^s(t)(f_{i,j}^s(t)-f_{j,i}^s(t)) +2\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}U_i^s(t)A_s(t)1_{[i=o(s)]}|\boldsymbol{U}(t)\Bigg]. \tag{11}$$

When we insert (10) in (11), we have

$$\Delta(\boldsymbol{U}(t))\leq B+2\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}U_i^s(t)\lambda_i^s -2E\Bigg[\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}U_i^s(t)\left(\sum_{j\in\mathcal{N}}f_{i,j}^{*\,s}(t)-\sum_{j\in\mathcal{N}}f_{j,i}^{*\,s}(t)\right)|\boldsymbol{U}(t)] +2|\mathcal{L}||\mathcal{S}|KF_{\max}. \tag{12}$$

If the vector of arrival rates are interior to the stability region, there always exist $\epsilon>0$ such that $E[\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}$ $(\sum_{j\in\mathcal{N}}f_{j,i}^{*\,s}(t)-\sum_{j\in\mathcal{N}}f_{i,j}^{*\,s}(t))|\boldsymbol{U}(t)]\leq-(\lambda_s 1_{[i=o(s)]}+\epsilon)$. Substituting this into (12);

$$\Delta(\boldsymbol{U}(t))\leq B+2|\mathcal{L}||\mathcal{S}|KF_{\max}-2\sum_{i\in\mathcal{N}}\sum_{s\in\mathcal{S}}U_i^s(t)\epsilon. \tag{13}$$

The time average of (13) yields;

$$\limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\sum_{s\in\mathcal{S}}U_i^s(\tau)\leq\frac{B+2|\mathcal{L}||\mathcal{S}|KF_{\max}}{2\epsilon}, \tag{14}$$

which shows that the time average of the sum of $U_i^s(t)$ over all flows is bounded. This conclude that TCP-aware backpressure stabilizes the network and the total average backlog is bounded.

APPENDIX B
STEADY STATE THROUGHPUT OF TCP WITH TCP-AWARE BACKPRESSURE

We consider the interaction of TCP-aware backpressure with TCP, assuming that TCP-Reno is employed.

Let the TCP congestion window size of flow s at its source node $o(s)$ and at time t is $W_{o(s)}(t)$. We assume that the RTT of each packet is constant, and T_s. This is a common assumption in classical TCP analysis [17], [18]. It is also a valid assumption in our system thanks to employing network coding. Since network coding makes packets sequence agnostic, we do not look at the RTT of each individual packet, rather the RTT of stream of packets. In particular, even if a packet takes a longer path, it does not increase RTT of this packet, because the ACK of this packet will be mapped to packet which is transmitted later. This reduces the jitter of RTT, and makes constant RTT assumption valid.

Let $q_i^s(t)$ be the probability that packets are dropped from buffers due to overflow. Thanks to employing backpressure for scheduling packets, we do not expect buffer overflow in the nodes except at the edge/source node. I.e., $q_i^s(t)=0$, $\forall i\in\mathcal{N}-\{o(s)\}$. Let $\rho_i^s(t)$ is the probability that packets from flow s could be transmitted from node i according to the underlying scheduling algorithm, i.e., $\rho_i^s(t)$ is the probability that packets are not trapped at node i.

At time $t-T_s$, $W_{o(s)}(t-T_s)$ packets are transmitted from TCP source $o(s)$. The ACKs corresponding to these packets, received between t and $t+T_s$, determine window size update. In particular, for each transmitted and ACKed packets, window size is increased by $\frac{1}{W_{o(s)}(t)}$ according to TCP-Reno. Therefore, the total increase in window size from time t to $t+T_s$ is; $W_{o(s)}(t-T_s)\frac{1}{W_{o(s)}(t)}(1-q_{o(s)}^s(t))$ $\prod_{i\in\mathcal{N}_s}\rho_i^s(t)$, where successful ACKs are received if packets are not dropped due to buffer overflow (with probability $(1-q_{o(s)}^s(t))$) and if each node in the network could forward packets using the underlying routing and scheduling algorithm (with probability $\rho_i^s(t)$). (Note that $\mathcal{N}_s$ is the set of nodes that are able to forward packets from flow s towards

their destination, and we assume that packets from flow s are routed to node i only if $i \in \mathcal{N}_s$).

For each dropped packet due to buffer overflow or for trapped packets in the buffers, window size is reduced in TCP-Reno by $W_{o(s)}(t)\beta$, where $0 < \beta < 1$. The total decrease from time t to $t + T_s$ is; $W_{o(s)}(t - T_s)\beta W_{o(s)}(t)$ $\{1 - (1 - q^s_{o(s)}(t)) \prod_{i\in\mathcal{N}_s} \rho^s_i(t)\}$. Thus, the window size evolution is expressed as

$$W_{o(s)}(t+T_s) = W_{o(s)}(t) + W_{o(s)}(t-T_s)\frac{1}{W_{o(s)}(t)}(1-q^s_{o(s)}(t)) \prod_{i\in\mathcal{N}_s}\rho^s_i(t) - W_{o(s)}(t-T_s)\beta W_{o(s)}(t) \left\{1-(1-q^s_{o(s)}(t))\prod_{i\in\mathcal{N}_s}\rho^s_i(t)\right\}. \tag{15}$$

Noting that $\dot{W}_{o(s)}(t) = \frac{W_{o(s)}(t+T_s)-W_{o(s)}(t)}{T_s}$ and $x_s(t) = \frac{W_{o(s)}(t)}{T_s}$ as well as considering that $\dot{W}_{o(s)} = 0$ in the steady state, x_s is expressed as

$$x_s^2 = \frac{(1-q^s_{o(s)})\prod_{i\in\mathcal{N}_s}\rho^s_i}{T_s^3\beta(1-(1-q^s_{o(s)})\prod_{i\in\mathcal{N}_s}\rho^s_i(t))}. \tag{16}$$

As it is seen from (16), TCP rate x_s depends on buffer overflow probability at the source node, i.e., $q^s_{o(s)}$, and transmission probability at each intermediate node $i \in \mathcal{N}_s$, i.e., ρ^s_i. Thus, in the classical backpressure either ρ^s_i information should be known by the source node to arrange the window size (and rate), which is not practical, or neglect this information, which is detrimental as ρ^s_i may receive very small values in classical backpressure, which reduces x_s. TCP-aware backpressure, on the other hand, increases ρ^s_i thanks to giving transmission opportunity to small flows according to (2). In practice, K is selected as $K = Q_i/|\mathcal{S}_i|$ as explained in Section 4.1.4. This significantly increases ρ^s_i. We have observed through our simulations in Section 5 that, very few packets are trapped in the buffers in TCP-aware backpressure. Such packets are easily masked thanks to error correcting capabilities of network coding. Therefore, in TCP-aware backpressure, $\rho^s_i \approx 1$. Note that ρ^s_i is the probability of not trapping packets at node i from flow s in a fluid-based model. Since TCP-aware backpressure almost eliminates the packet trapping probability, ρ^s_i is close to 1. Thus, (16) is expressed as

$$x_s^2 = \frac{(1-q^s_{o(s)})}{T_s^3\beta q^s_{o(s)}}, \tag{17}$$

where x_s only depends only on the buffer overflow probability at the source/edge node.

APPENDIX C
OPTIMALITY ANALYSIS OF TCP-AWARE BACKPRESSURE WITH REAL TCP FLOWS

The Karush-Kuhn-Tucker (KKT) optimality conditions of the optimization problem:

$$\begin{aligned} \max_{\boldsymbol{x},\boldsymbol{f}} \; & \sum_{s\in\mathcal{S}} g_s(x_s) \\ \text{s.t.} \; & \sum_{j\in\mathcal{N}} f^s_{i,j} - \sum_{j\in\mathcal{N}} f^s_{j,i} = x_s 1_{[i=o(s)]}, \forall i \in \mathcal{N}, s \in \mathcal{S} \\ & \boldsymbol{f} \in \Gamma, \end{aligned} \tag{18}$$

are

$$\begin{aligned} & q^s_i\left(\sum_{j\in\mathcal{N}} f^s_{i,j} - \sum_{j\in\mathcal{N}} f^s_{j,i} - x_s 1_{[i=o(s)]} = 0\right), \forall i \in \mathcal{N}, s \in \mathcal{S} \\ & g'_s(x_s) - q^s_{o(s)} = 0, \forall s \in \mathcal{S}, \end{aligned} \tag{19}$$

where q^s_i are KKT multipliers and they correspond to packet dropping probability from flow s due to buffer overflow at node i. Next, we show that our TCP-aware backpressure algorithm with TCP-Reno sources satisfy the KKT optimality conditions in (19).

Let us consider the first KKT condition in (19). In our algorithm, if $\sum_{j\in\mathcal{N}} f^s_{i,j} - \sum_{j\in\mathcal{N}} f^s_{j,i} - x_s 1_{[i=o(s)]} > 0$, then the arrival rate is smaller than the departure rate, and $q^s_i = 0$ in the equilibrium since we are using the fluid model. Thus, the first KKT condition is satisfied in this case.

On the other hand, we cannot possibly have $\sum_{j\in\mathcal{N}} f^s_{i,j} - \sum_{j\in\mathcal{N}} f^s_{j,i} - x_s 1_{[i=o(s)]} < 0$ if $\boldsymbol{f} \in \Gamma$ in the equilibrium, because TCP-Reno sources do not introduce inadmissible rates. Thus, we cannot have $\sum_{j\in\mathcal{N}} f^s_{i,j} - \sum_{j\in\mathcal{N}} f^s_{j,i} - x_s 1_{[i=o(s)]} < 0$, and if $q^s_i \neq 0$, we should have $\sum_{j\in\mathcal{N}} f^s_{i,j} - \sum_{j\in\mathcal{N}} f^s_{j,i} - x_s 1_{[i=o(s)]} = 0$. Thus, the first KKT condition is always satisfied.

Now, let us consider the second KKT condition in (19). In this step, we use the equilibrium analysis of TCP provided in Appendix B. In particular, in Appendix B, we have shown that the window size evolution is expressed as in (15). Noting that the window size and TCP rates are related to each other according to $x_s(t) = \frac{W_{o(s)}(t)}{T_s}$, we have

$$\frac{x_s(t+T_s)-x_s(t)}{T_s} = \frac{x_s(t-T_s)}{T_s^2 x_s(t)}(1-q^s_{o(s)}(t)) \prod_{i\in\mathcal{N}_s}\rho^s_i(t) - \beta x_s(t-T_s)x_s(t) \left\{1-(1-q^s_{o(s)}(t))\prod_{i\in\mathcal{N}_s}\rho^s_i(t)\right\}. \tag{20}$$

In the equilibrium, (20) is expressed as

$$\dot{x}_s = \frac{(1-q^s_{o(s)})\prod_{i\in\mathcal{N}_s}\rho^s_i}{T_s^2} - \beta x_s^2\left\{1-(1-q^s_{o(s)})\prod_{i\in\mathcal{N}_s}\rho^s_i\right\}. \tag{21}$$

Since packet trapping probability is very low in our algorithm, $\prod_{i\in\mathcal{N}_s}\rho^s_i$ is very close to 1. Thus, we have

$$\begin{aligned} \dot{x}_s &= \frac{(1-q^s_{o(s)})}{T_s^2} - \beta x_s^2 q^s_{o(s)} \\ &= \left(\beta x_s^2 + \frac{1}{T_s^2}\right)\left(\frac{1}{\beta x_s^2 + 1/T_s^2}\frac{1}{T_s^2} - q^s_{o(s)}\right). \end{aligned} \tag{22}$$

On the other hand, the second KKT condition, i.e., $g_s'(x_s) - q^s_{o(s)} = 0$ gives the same set of solutions as $\dot{x}_s = k_s(x_s)\ (g_s'(x_s) - q^s_{o(s)})$ by setting $\dot{x}_s = 0$ if (i) $k_s(x_s)$ is a step size parameter and (ii) $k_s(x_s) > 0$. Thus, according to TCP rate analysis in (22), the second KKT condition is satisfied for $k_s(x_s) = \beta x_s^2 + 1/T_s^2$ and $g_s'(x_s) = \frac{1}{T_s^2}\frac{1}{\beta x_s^2 + 1/T_s^2}$, where $g_s(x_s)$ can be expressed as $g_s(x_s) = \frac{1}{T_s\sqrt{\beta}} \tan^{-1}(\sqrt{\beta} T_s x_s)$.

Now that we showed that TCP-aware backpressure with TCP-Reno sources satisfy the KKT optimality conditions in the equilibrium, we can conclude that our TCP-aware backpressure algorithm with real TCP-Reno sources approaches to the solution of (18) in the equilibrium.

ACKNOWLEDGMENTS

This work was supported by the US National Science Foundation (NSF) grant CNS-1217048, ONR grant N00014-12-1-0064, and ARO Muri grant number W911NF-08-1-0238. The preliminary results of this paper were presented in part at the IEEE Information Theory and Applications Workshop, San Diego, CA, Feb. 2014.

REFERENCES

[1] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Trans. Automat. Control*, vol. 37, no. 12, pp. 1936–1948, Dec. 1992.

[2] L. Tassiulas and A. Ephremides, "Dynamic server allocation to parallel queues with randomly varying connectivity," *IEEE Trans. Inf. Theory*, vol. 39, no. 2, pp. 466–478, Mar. 1993.

[3] M. J. Neely, E. Modiano, and C. Li, "Fairness and optimal stochastic control for heterogeneous networks," *IEEE/ACM Trans. Netw.*, vol. 16, no. 2, pp. 396–409, Apr. 2008.

[4] K. Tan, J. Song, Q. Zhang, and M. Sridharan,"A compound TCP approach for high-speed and long distance networks," in *Proc. 25th IEEE Int. Conf. Comput. Commun. INFOCOM*, Barcelona, Spain, Apr. 2006, pp. 1–12.

[5] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008.

[6] The Network Simulator - ns-2, Version 2.35 [Online]. Available: www.isi.edu/nsnam/ns/, Nov. 2005.

[7] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*, San Rafael, CA, USA: Morgan & Claypool, 2010.

[8] Y. Yi, A. Proutiere, and M. Chiang, "Complexity in wireless scheduling: Impact and tradeoffs," in *Proc. 9th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, Hong Kong, China, May 2008, pp. 33–42.

[9] M. Chiang, S. T. Low, A. R. Calderbank, and J. C. Doyle, "Layering as optimization decomposition: A mathematical theory of network architectures," *Proc. IEEE*, vol. 95, no. 1, pp. 255–312, Jan. 2007.

[10] X. Lin, N. B. Schroff, and R. Srikant, "A tutorial on cross-layer optimization in wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 8, pp. 1452–1463, Aug. 2006.

[11] H. Seferoglu and E. Modiano, "Diff-Max: Separation of routing and scheduling in backpressure-based wireless networks," in *Proc. IEEE INFOFOCM*, Turin, Italy, Apr. 2013, pp. 1555–1563.

[12] L. X. Bui, R. Srikant, and A. Stolyar, "A novel architecture for reduction of delay and queueing structure complexity in the backpressure algorithm," *IEEE/ACM Trans. Netw.*, vol. 19, no. 6, pp. 1597–1609, Dec. 2011.

[13] J. K. Sundararajan, D. Shah, M. Medard, M. Mitzenmacher, and J. Barros, "Network coding meets TCP," in *Proc. IEEE INFOCOM*, Rio de Janeiro, Brazil, Apr. 2009, pp. 280–288.

[14] S. Gheorghiu, A. L. Toledo, and P. Rodriguez, "Multi-path TCP with network coding for wireless mesh networks," in *Proc. IEEE Int. Conf. Commun.*, Cape Town, South Africa, May 2010, pp. 1–5.

[15] H. Seferoglu, A. Markopoulou, and K. K. Ramakrishnan, "I^{2}NC: Intra- and inter-session network coding for unicast flows in wireless networks," in *Proc. IEEE INFOCOM*, Shanghai, China, Apr. 2011, pp. 1035–1043.

[16] P. A. Chou and Y. Wu, "Network coding for the internet and wireless networks," *IEEE Signal Proc. Mag.*, vol. 24, no. 5, pp. 75–85, Sep. 2007.

[17] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. ACM SIGCOMM*, Vancouver, BC, Canada, Sep. 1998, pp. 303–314.

[18] S. Low, "A duality model of TCP and queue management algorithms," *IEEE/ACM Trans. Netw.*, vol. 11, no. 4, pp. 525–536, Aug. 2003.

[19] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (AODV) routing," *RFC 3561, IETF*, Jul. 2003.

[20] R. K. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, Hoboken, NJ, USA: Wiley, Apr. 1991.

[21] A. L. Stolyar, "Greedy primal dual algorithm for dynamic resource allocation in complex networks," *Queuing Syst.*, vol. 54, no. 3, pp. 203–220, 2006.

[22] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali, "Routing without routes: The backpressure collection protocol," in *Proc. 9th ACM/IEEE Int. Process. Sensor Netw.*, Stockholm, Sweden, Apr. 2010, pp. 279–290.

[23] R. Laufer, T. Salonidis, H. Lundgren, and P. L. Guyadec, "XPRESS: A cross-layer backpressure architecture for wireless multi-hop networks," in *Proc. 17th Annu. Int. Conf. Mobile Comput. Netw.*, Las Vegas, NV, USA, Sep. 2011, pp. 49–60.

[24] A. A. Bhorkar, T. Javidi, and A. C. Snoereny, "Achieving congestion diversity in wireless ad-hoc networks," in *Proc. IEEE INFOCOM*, Shanghai, China, Apr. 2011, pp. 521–525.

[25] K. Choumas, T. Korakis, I. Koutsopoulos, and L. Tassiulas, "Implementation and end-to-end throughput evaluation of an IEEE 802.11 compliant version of the enhanced-backpressure algorithm," in *Proc. 8th Int. ICST Conf. Testbeds Res. Infrastructure. Development of Networks and Communities*, Thessaloniki, Greece, Jun. 2012, pp. 64–80.

[26] J. Ryu, V. Bhargava, N. Paine, and S. Shakkottai, "Backpressure routing and rate control for ICNs," in *Proc. ACM 16th Annu. Int. Conf. Mobile Comput. Netw.*, Chicago, IL, Sep. 2010, pp. 365–376.

[27] A. Warrier, S. Janakiraman, S. Ha, and I. Rhee, "DiffQ: Practical differential backlog congestion control for wireless networks," in *Proc. IEEE INFOCOM*, Rio de Janerio, Brazil, Apr. 2009, pp. 262–270.

[28] U. Akyol, M. Andrews, P. Gupta, J. Hobby, I. Saniee, and A. Stolyar, "Joint scheduling and congestion control in mobile ad-hoc networks," presented at the IEEE 27th Conf. Comput. Commun., Phoenix, AZ, USA, Apr. 2008.

[29] B. Radunovic, C. Gkantsidis, D. Gunawardena, and P. Key, "Horizon: Balancing TCP over multiple paths in wireless mesh network," in *Proc. 14th ACM Int. Conf. Mobile Comput. Netw.*, San Francisco, CA, Sep. 2008, pp. 247–258.

[30] J. Ghaderi, T. Ji, and R. Srikant,, "Connection-level scheduling in wireless networks using only MAC-layer information," in *Proc. IEEE INFOCOM*, Orlando, FL, USA, Mar. 2012, pp. 2696–2700.

[31] A. Shpiner and I. Keslassy, "Modeling the interactions of congestion control and switch scheduling," *Comput. Netw.*, vol. 55, no. 6, pp. 1–9, Apr. 2011.

[32] P. Giaccone, E. Leonardi, and F. Neri, "On the interaction between TCP-like sources and throughput-efficient scheduling policies," *J. Perform. Eval.*, vol. 70, no. 4, pp. 251–270, 2013.

[33] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. Conf. Applicat, Tech., Archit., Protocol Comput. Commun.*, Aug. 2000, pp. 43–56.

[34] R. Srikant and L. Ying, *Communication Networks: An Optimization, Control, and Stochastic Networks Perspective*. Cambridge, U.K. Academic Press, Feb. 2014.

[35] B. Ji, C. Joo, and N. B. Shroff, "Delay-based back-pressure scheduling in multihop wireless networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, pp. 1539–1552, Oct. 2013.

Hulya Seferoglu (S'04-M'11) received the BS degree in electrical engineering from Istanbul University, Turkey, in 2003, the MS degree in electrical engineering and computer science from Sabanci University, Turkey, in 2005, and the PhD degree in electrical and computer engineering from the University of California, Irvine, in 2010. She is an assistant professor in the Electrical and Computer Engineering Department, University of Illinois at Chicago. She was a postdoctoral associate in the Laboratory of Information and Decision Systems (LIDS), Massachusetts Institute of Technology during 2011-2013. She was a summer intern at AT&T Labs Research, Docomo USA Labs, and Microsoft Research Cambridge in 2010, 2008, and 2007, respectively. Her research interests are in the area of networking: design, analysis, and optimization of network protocols and algorithms. She is particularly interested in network optimization, network coding, and multimedia streaming. She is a member of the IEEE.

Eytan Modiano (F'12) received the BS degree in electrical engineering and computer science from the University of Connecticut at Storrs in 1986 and the MS and PhD degrees, both in electrical engineering, from the University of Maryland, College Park, MD, in 1989 and 1992, respectively. He was a naval research laboratory fellow between 1987 and 1992, and a national research council post doctoral fellow during 1992-1993. Between 1993 and 1999, he was with MIT Lincoln Laboratory, where he was a project leader for MIT Lincoln Laboratory's Next Generation Internet (NGI) project. Since 1999, he has been on the faculty at MIT, where he is a professor in the Department of Aeronautics and Astronautics and the Laboratory for Information and Decision Systems (LIDS). His research is on communication networks and protocols with emphasis on satellite, wireless, and optical networks. He is an editor-at-large for the *IEEE/ACM Transactions on Networking*, and served as an associate editor for the *IEEE Transactions on Information Theory* and *IEEE/ACM Transactions on Networking*. He was the technical program co-chair for IEEE Wiopt 2006, IEEE Infocom 2007, and ACM MobiHoc 2007. He is a fellow of the IEEE and an associate fellow of the AIAA.